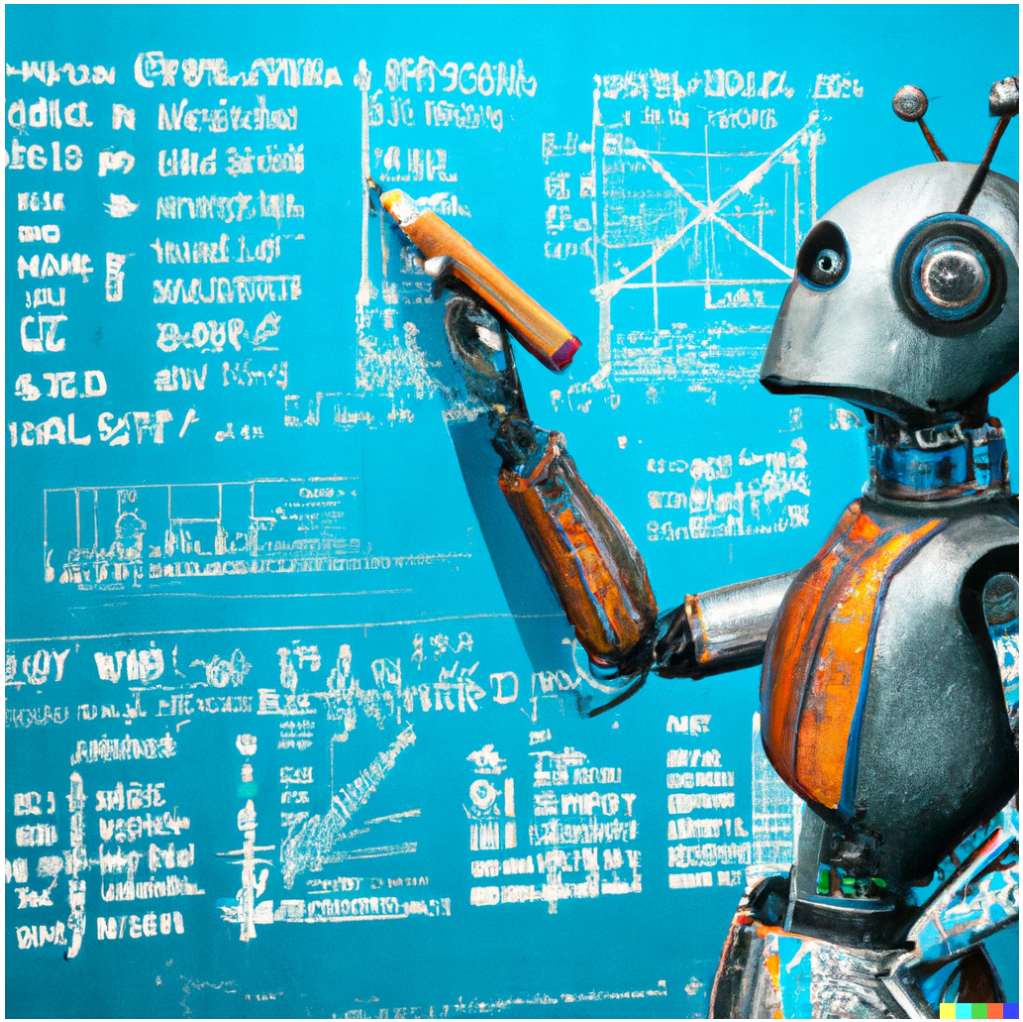
Introduction to Deep Learning

Elizaveta Semenova

Imperial College London
19 March 2024

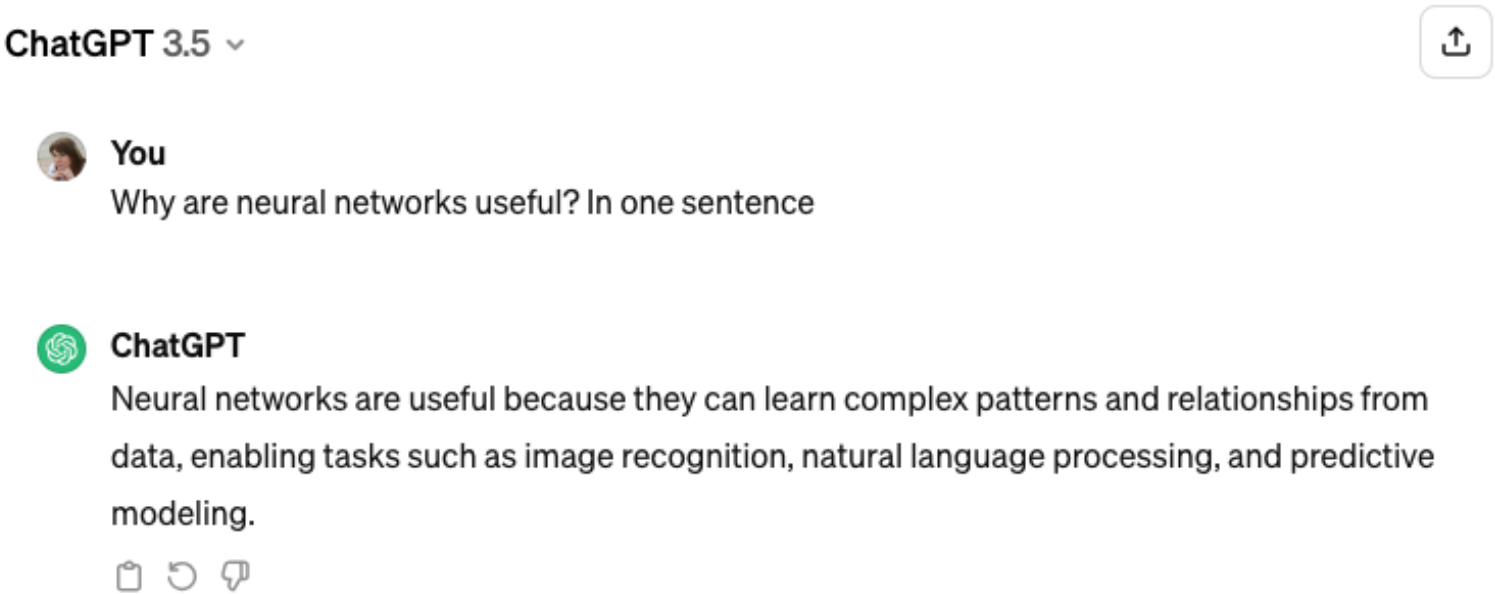
Outline

Modern applications of machine learning



"Robot mixing statistics, calculus and linear algebra, digital art", DALLE-2

Modern applications of machine learning



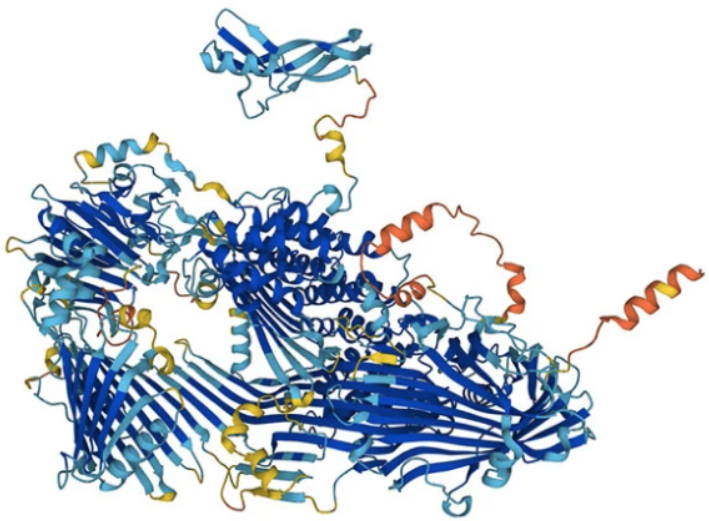
Modern applications of machine learning

NEWS | 18 January 2024

AlphaFold found thousands of possible psychedelics. Will its predictions help drug discovery?

Researchers have doubted how useful the AI protein-structure tool will be in discovering medicines – now they are learning how to deploy it effectively.

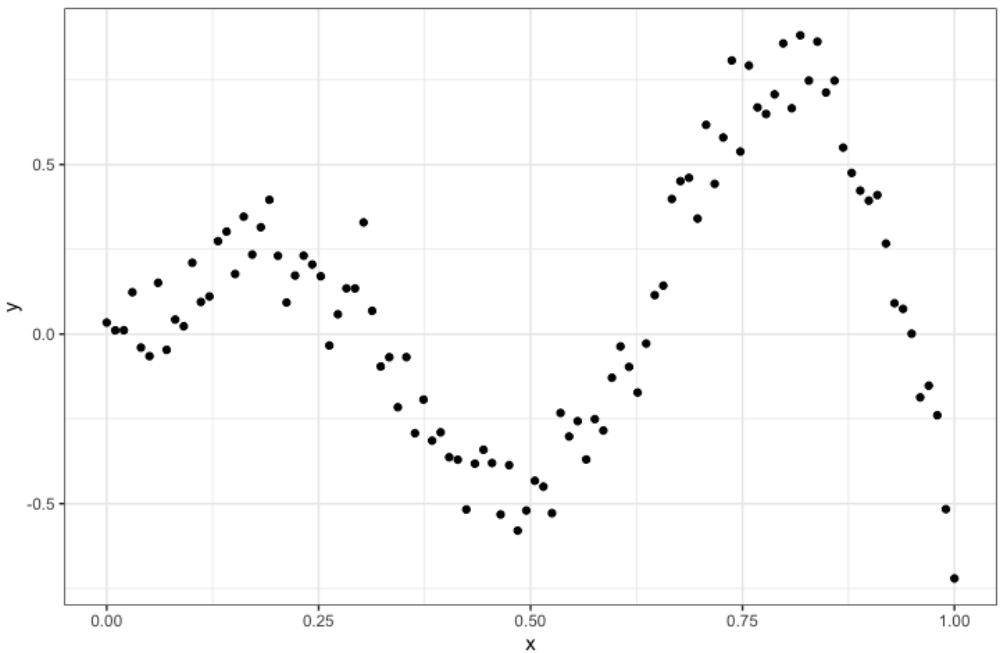
By Ewen Callaway



Protein structures predicted by AlphaFold have helped to identify candidate drug compounds. Credit: DeepMind

Curve fitting

- Problem setting: given observed data pairs $(x_i, y_i), i = 1, ..., n$, learn to predict y from x.



Curve fitting

- Problem setting: given observed data pairs $(x_i, y_i), i = 1, ..., n$, learn to predict y from x.
- Model f, depending on parameters θ: $y = f(x, \theta)$.

Curve fitting

- Problem setting: given observed data pairs $(x_i, y_i), i = 1, ..., n$, learn to predict y from x.

- Model f, depending on parameters θ: $y = f(x, \theta)$.

linear model	$f(x, \theta) = \theta^T x$	parametric, linear
Gaussian process	$f(x, \theta) = GP_\theta(x)$	nonparametric
neural network	$f(x, \theta) = NN_\theta(x, \theta)$	parametric, non-linear

Curve fitting

- Problem setting: given observed data pairs $(x_i, y_i), i = 1, ..., n$, learn to predict y from x.
- Model f, depending on parameters θ: $y = f(x, \theta)$.
- Training a model, means finding parameters θ^*, such that $f(x_i, \theta^*) \approx y_i$, e.g.:

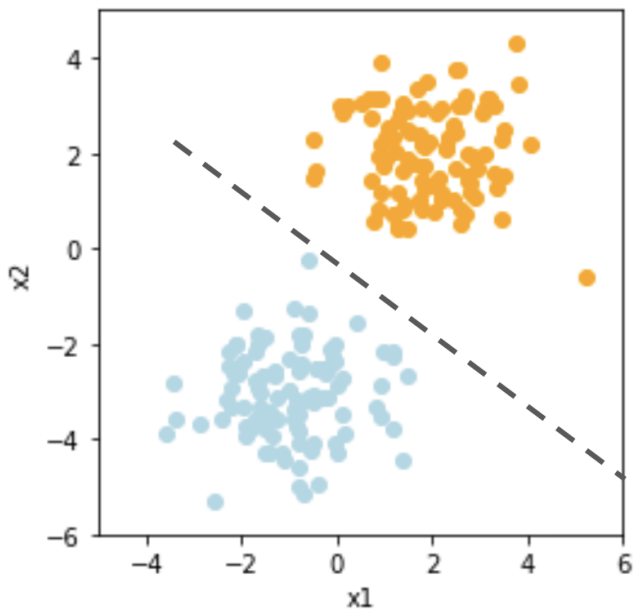
$$\sum_{i=1}^{n} (f(x_i, \theta^*) - y_i)^2 \rightarrow \min_{\theta}$$

Curve fitting

$$\underbrace{\sum_{i=1}^{n} \left(f(x_i, \theta^*) - y_i\right)^2}_{\mathcal{L}(x,y,\theta) \text{ - loss function}} \to \min_{\theta}$$

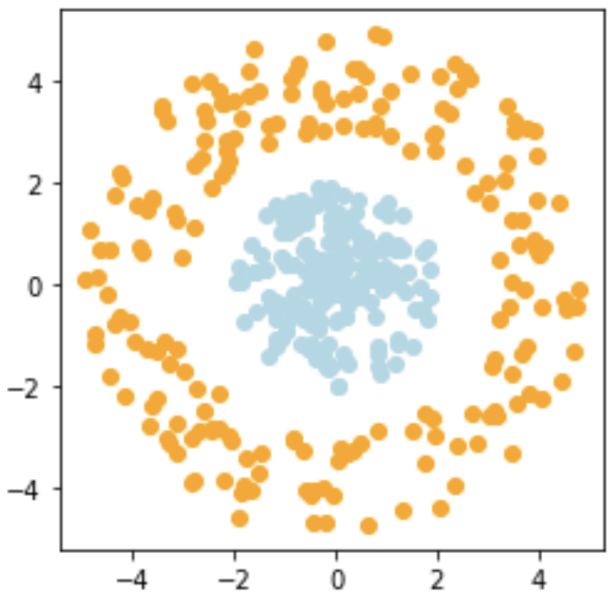
Classification problem

- Logistic regression is a **classification** model.
- It allows to separate two classes using a **linear** decision boundary.



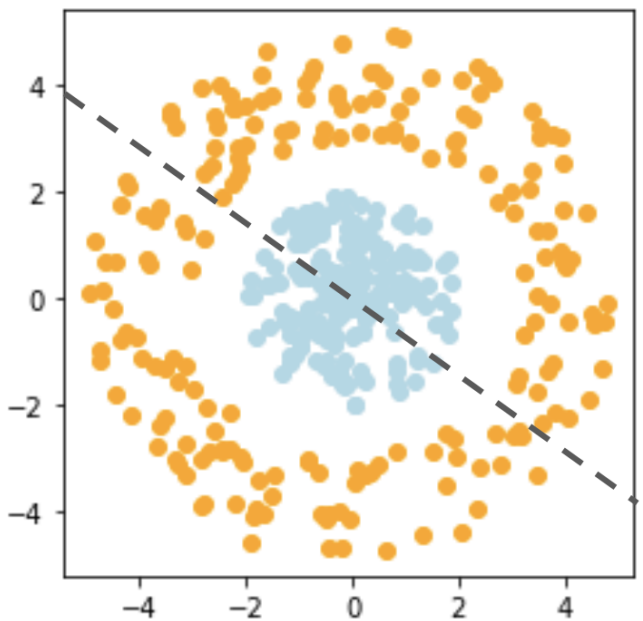
Classification problem

- What would happen if we apply logistic regression to this data?



Classification problem

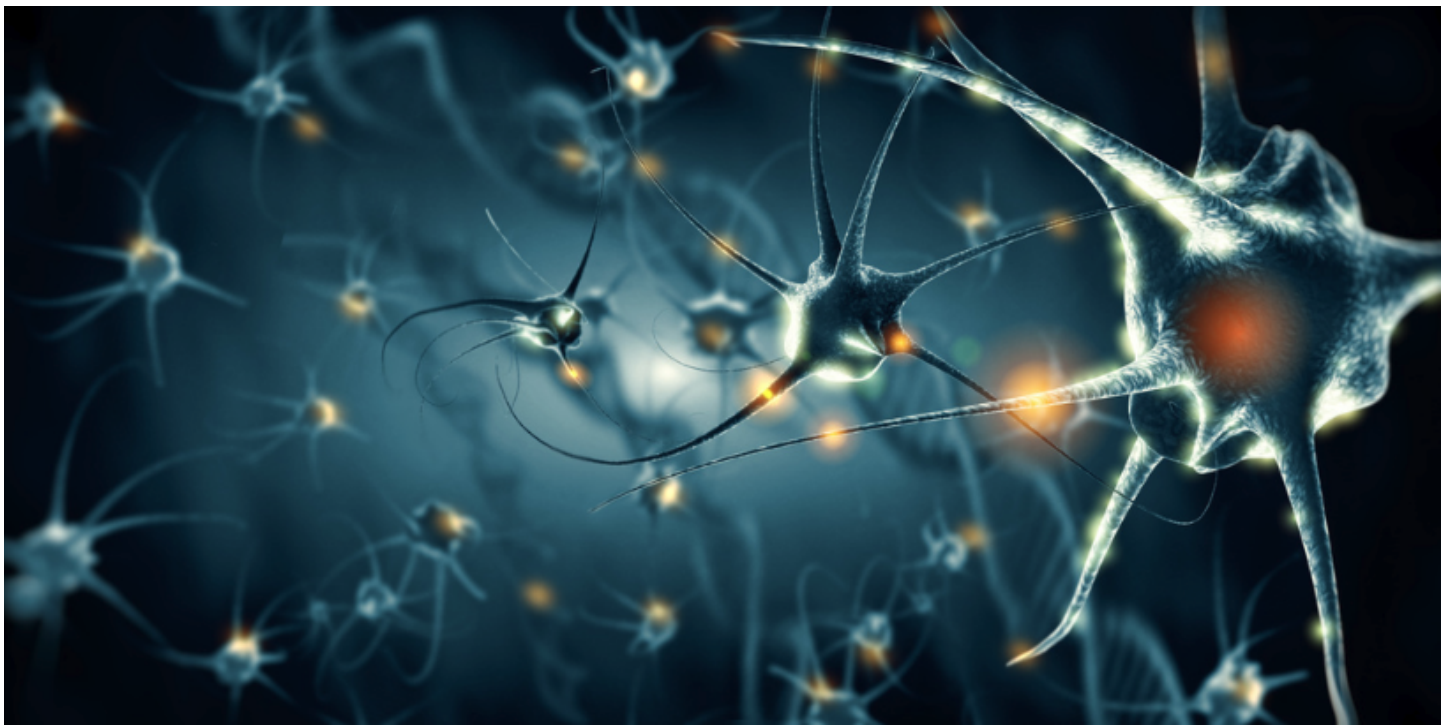
- What would happen if we apply logistic regression to this data?
- Not possible, the task is too complex!



Outline

Neurons in a brain

- The brain uses billions of interconnected neurons.
- Many simple interactions can solve complex tasks.
- Can we use this concept to make computers learn?

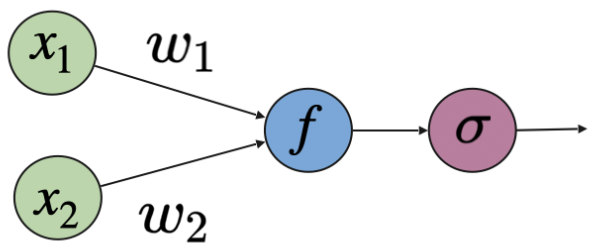


Computational model of a single neuron

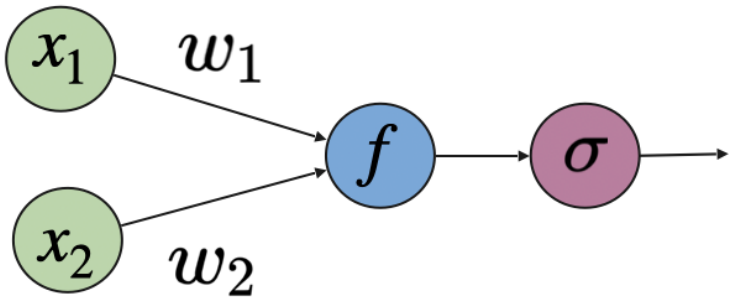
- Inputs x_1, x_2 are multiplied with **weights** w_1, w_2 and added together, and a **bias** b is added:

$$f(x_1, x_2) = x_1 * w_1 + x_2 * w_2 + 1 * b$$

- We then apply an **activation function**.



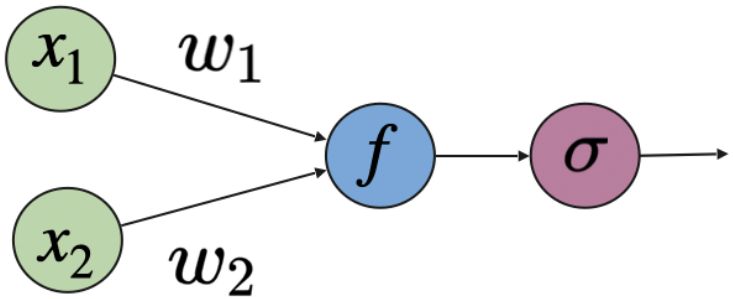
Computational model of a single neuron



$$f(x_1, x_2) = x_1 w_1 + x_2 w_2 + b,$$
$$p(x_1, x_2) = \sigma(f(x_1, x_2)),$$
$$\sigma(t) = \frac{1}{1 + \exp^{-t}}$$

- Does it remind you of anything?

Computational model of a single neuron



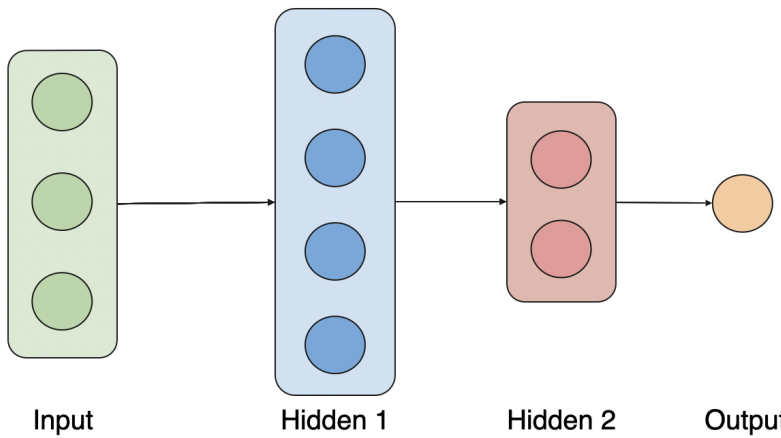
$$f(x_1, x_2) = x_1 w_1 + x_2 w_2 + b,$$
$$p(x_1, x_2) = \sigma(f(x_1, x_2)),$$
$$\sigma(t) = \frac{1}{1 + \exp^{-t}}$$

- Does it remind you of anything?
- Logistic regression!

Outline

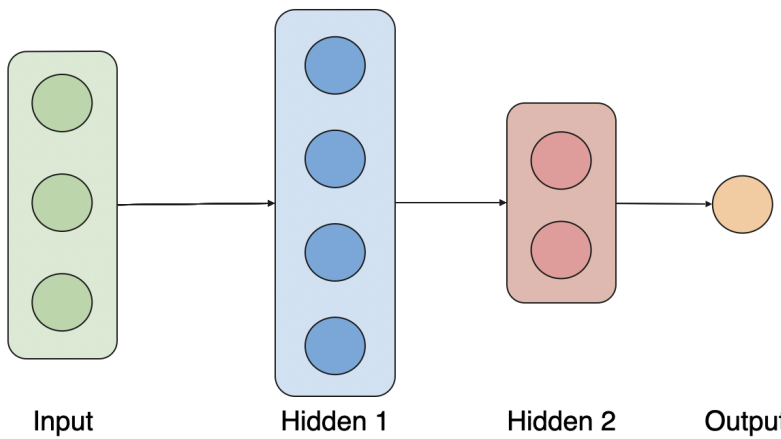
A neural network

- A neural network connects many individual neurons (**nodes**) together to solve complex tasks.
- The **architecture** of a neural network tells us how these connections should look like.



A neural network

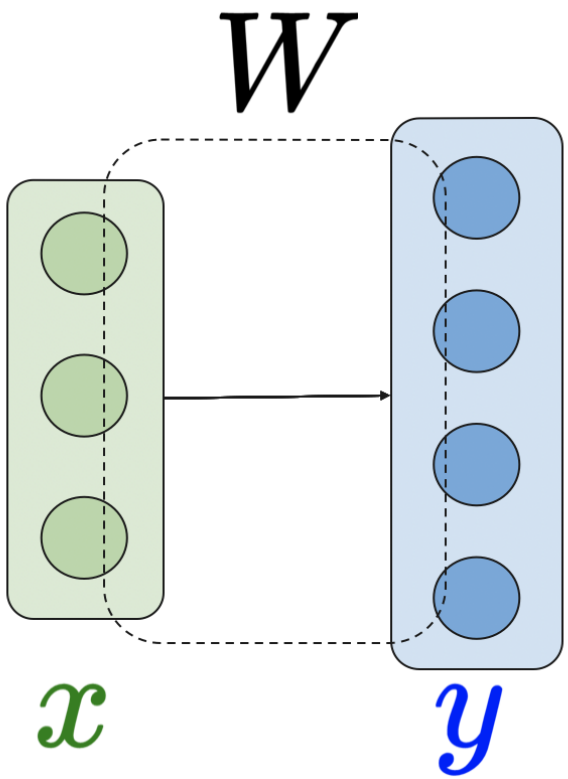
- The **input layer** consists of data - predictors, also called features.
- The **output layer** is the observed variable of interest.
- In-between are the **hidden layers**.



Mathematical foundations of deep learning

- How to derive an appropriate loss function for each problem?
  $\rightarrow$ Statistics
- How to optimise a loss function?
  $\rightarrow$ Calculus
- How to express computations efficiently?
  $\rightarrow$ Linear algebra

Neural networks in matrix notation

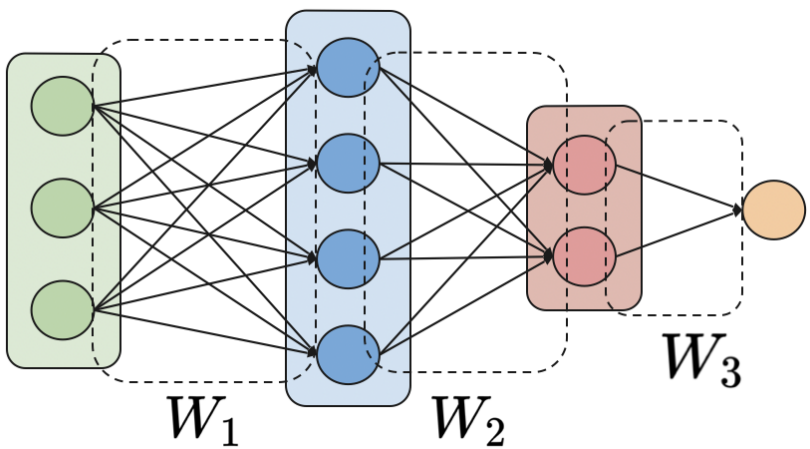


$$y = Wx$$

Network design

We refer to the design of the network as the **network architecture**:

- The number of hidden layers,
- The number of neurons per layer (referred to as width),
- The activation function for each neuron.

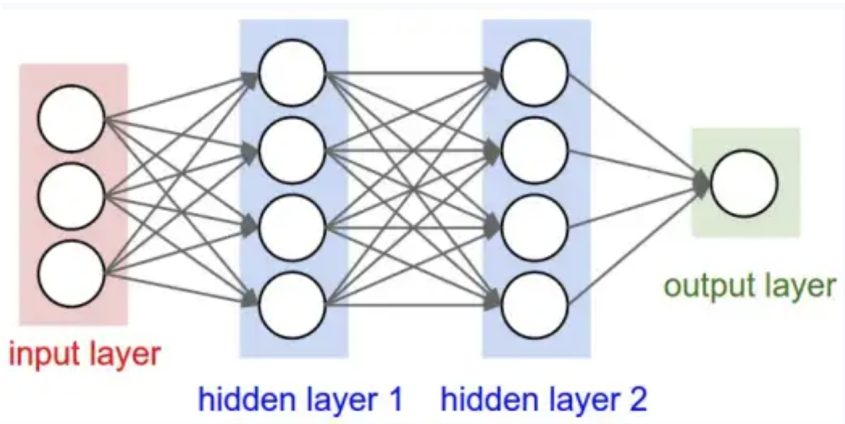


Architectures

The list of useful architectures is evolving daily as they are being discovered by trial-and-error.

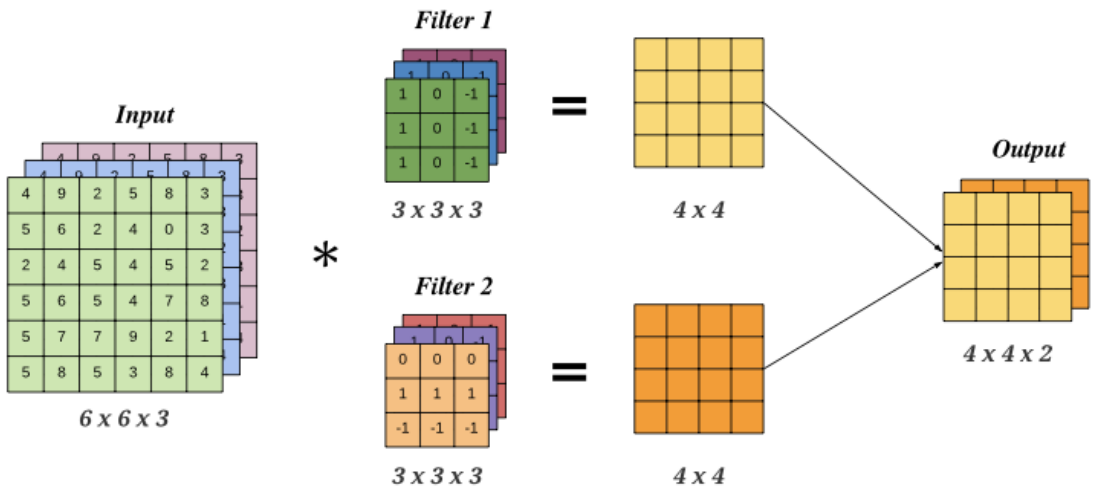
Multilayer perceptron (MLP)

- A **multilayer perceptron** is a feedforward architecture, consisting of fully connected neurons with a nonlinear kind of activation function.



Convolutional neural networks (CNNs)

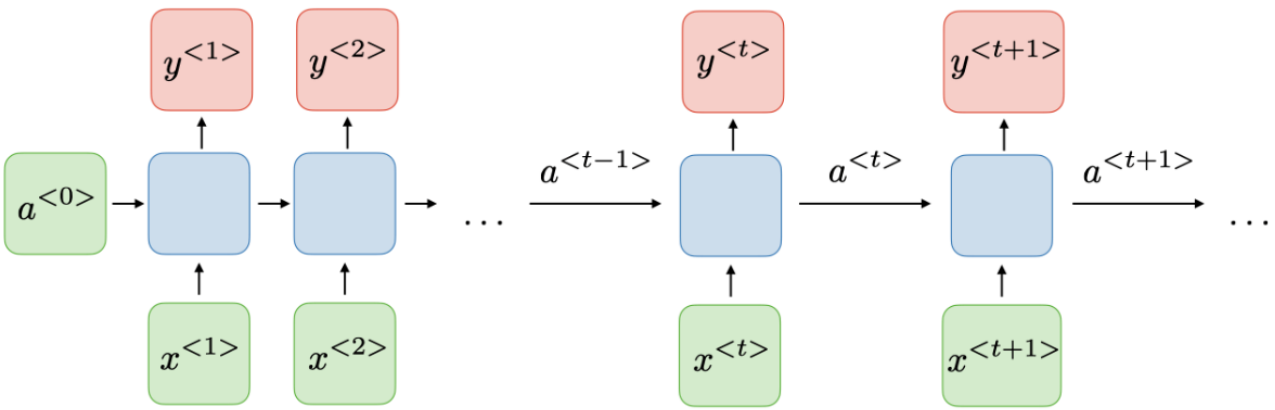
- CNNs use **convolution layers** which scan the input with a set of **filters**.
- **Pooling layers** are used for downsampling.



Credit: Kevin Murphy, "Probabilistic Machine Learning 2"

Recurrent neural networks (RNNs)

- RNNs allow previous outputs to be used as inputs while having hidden states. They are used to model **sequences**.



Credit: Shervine Amidi, Stanford CS230 course

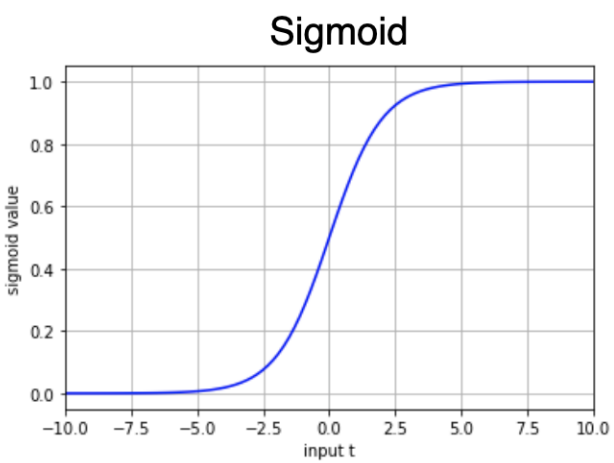
Activation functions

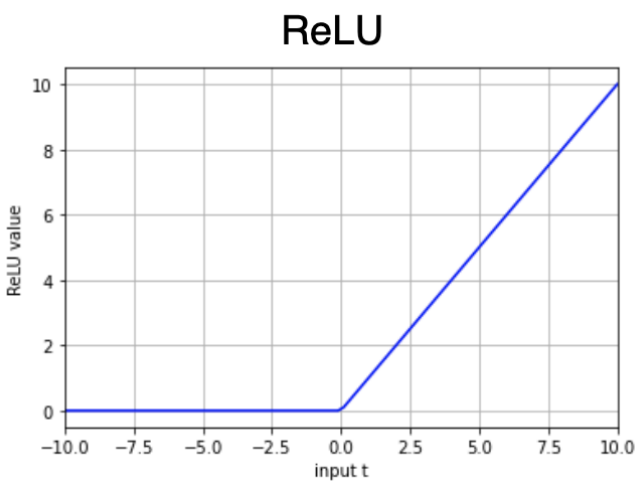
- Every node in the network has an **activation function**.
- To perform binary classification, our last node has a **sigmoid** activation

$$\sigma(t) = \frac{1}{1 + \exp -t}$$

- For nodes in the hidden layers we usually apply the Rectified Linear Unit (**ReLU**)

$$\mathrm{ReLU}(t) = \max(0, t)$$





Training a neural network

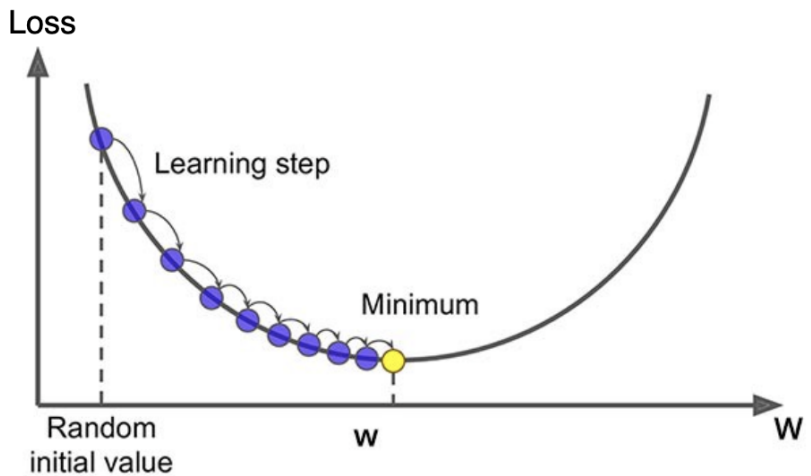
- **To train** a neural network means to find weights W that **minimise** the loss function $\mathcal{L}(W)$.

Gradient descent

- **Gradient descent** can be used to iteratively update the weights and get the best loss value

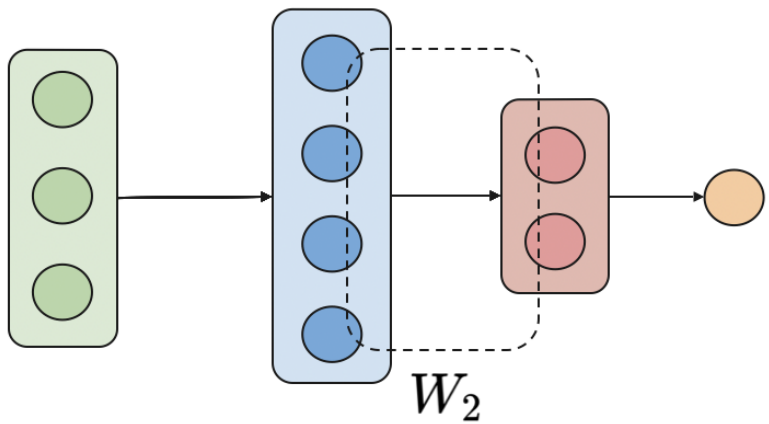
$$w_i := w_i - \alpha \frac{\partial \mathcal{L}}{\partial w_i}$$

- The step size is determined by the **learning rate** α.



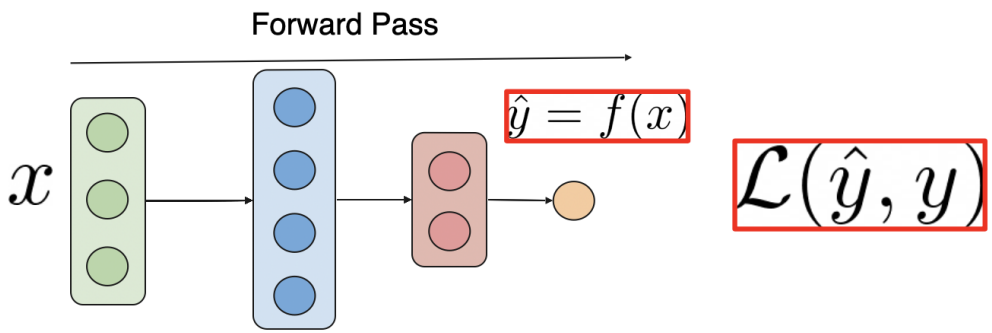
Back propagation

How can we update the hidden layers based on the loss value determined from the last layer?



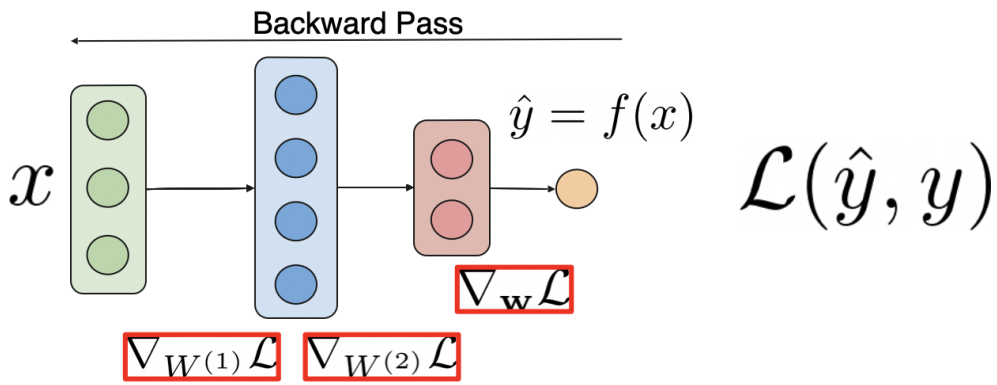
Forward pass

- Propagate inputs x through the network to compute the **predicted value** $\hat{y}$.
- Compute the **loss value** $\mathcal{L}(y, \hat{y})$ between the prediction $\hat{y}$ and the known ground-truth value y.



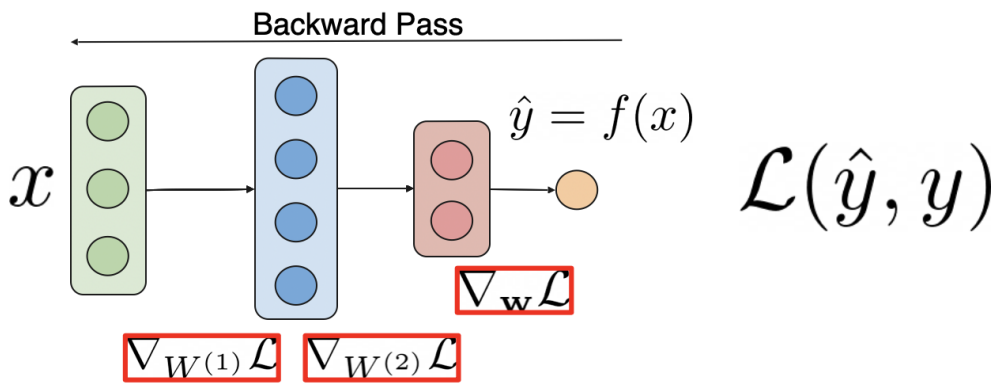
Backward pass

- Given the loss, compute the derivatives with respect to each layer ("back-propagation").
- Update the weights of each layer based on the computed gradients.



The training loop

- Define architecture.
- Load data.
- For a given number of iterations:
  - Compute model prediction $\hat{y}$,
  - Compute loss value $\mathcal{L}(y, \hat{y})$,
  - Perform back-propagation to obtain the gradients of the loss,
  - Update the networks parameters based on the gradient.



Outline

Deep learning tools: Python

Deep learning libraries: R

In R, there are some native libraries, as well as interfaces to libraries written in other languages:

- *neuralnet*,
- *deepnet*,
- *h2o*,
- *KerasR*

and more.

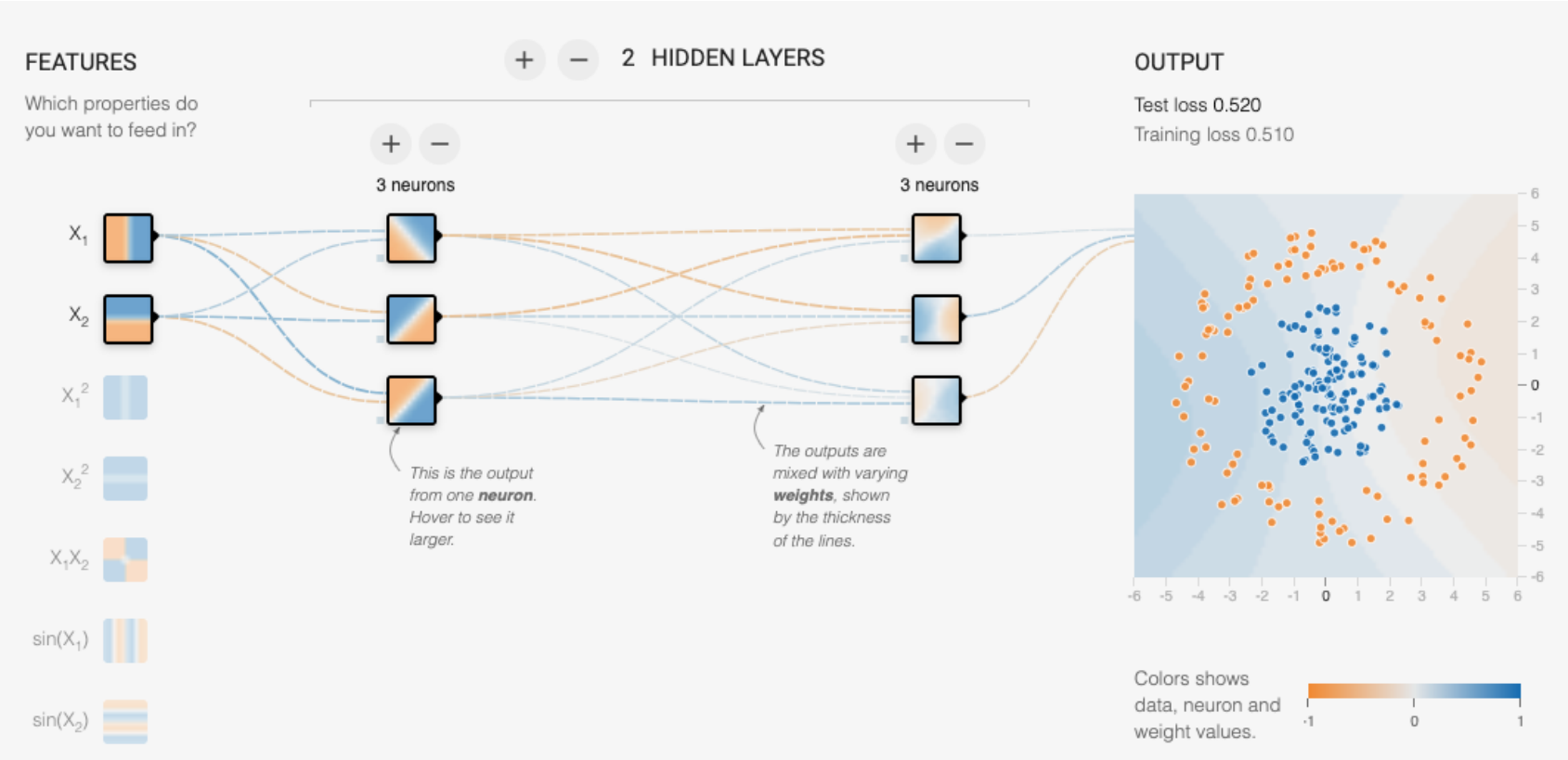
For examples, see

`https://srdas.github.io/DLBook/DeepLearningWithR.html`

Outline

Try for yourself

Search for 'tensorflow playground':

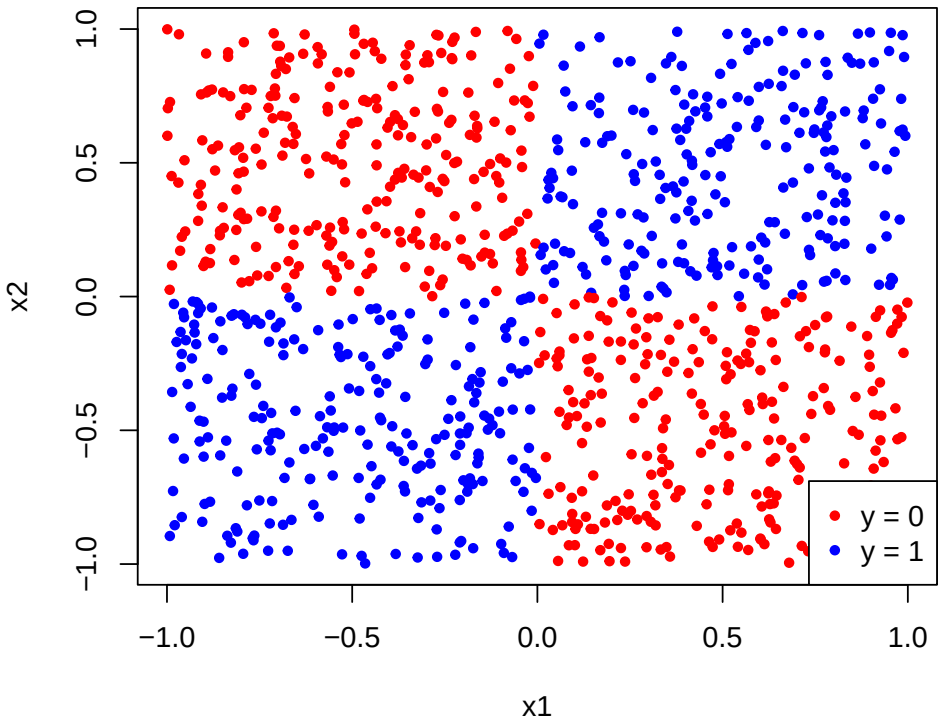


Example: binary classification

```r
# ----------------- simulate data -----------------

# Generate some sample data
set.seed(123)
n <- 1000
x1 <- runif(n, min = -1, max = 1)
x2 <- runif(n, min = -1, max = 1)
y <- ifelse((x1 * x2) > 0, 1, 0)  # binary classification task

# Visualize dependence between X and y
plot(x1, x2, col = ifelse(y == 1, "blue", "red"), pch = 20,
     #main = "Dependence between X and y",
     main = "",
     xlab = "x1", ylab = "x2")
legend("bottomright", legend = c("y = 0", "y = 1"), col = c("red", "blue"), pch = 20)

# Combine into a data frame
data <- data.frame(x1, x2, y)
```

Example: binary classification

Let's fit a binary classification model to this simulated dataset:



Example: define neural network

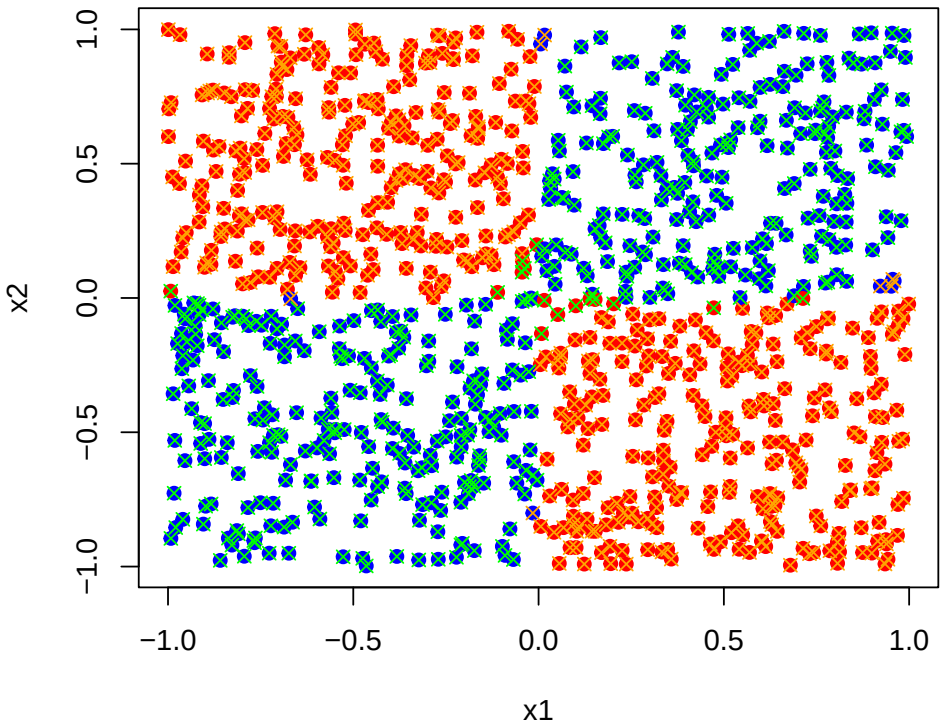
```r
# ----------------- define neural network -----------------

# Install and load required packages
library(neuralnet)


# Split data into training and testing sets
train_indices <- sample(1:n, 0.7 * n)
train_data <- data[train_indices, ]
test_data <- data[-train_indices, ]

# Define the neural network architecture
# Here, we have 2 input nodes (x1 and x2), 2 nodes in the hidden layer, and 1 output node
# We use 'sigmoid' as the activation function for both the hidden and output layers
nn <- neuralnet(y ~ x1 + x2, data = train_data, hidden = 5, act.fct = "logistic")
```

Example: evaluate neural network

```r
# ----------------- evaluate neural network -----------------

# Predict on test data
predicted <- predict(nn, test_data)

# Convert predicted probabilities to binary predictions
predicted_class <- ifelse(predicted > 0.5, 1, 0)

# Calculate accuracy
accuracy <- mean(predicted_class == test_data$y)
cat("Accuracy:", accuracy, "\n")

predicted_full <- predict(nn, data)
y_pred_full <- ifelse(predicted_full > 0.5, 1, 0)

# Visualize dependence between X and y
plot(x1, x2, col = ifelse(y == 1, "blue", "red"), pch = 19,
     main = "Dependence between X and y",
     xlab = "x1", ylab = "x2")
points(x1, x2, col = ifelse(y_pred_full == 1, "green", "orange"), pch = 4)
```

Example: evaluate neural network

How well does the model perform?



Example: code

This code is available at
`https://github.com/elizavetasemenova/nn_in_R`

How to learn more?

- "Dive into deep learning", Zhang, Smola, Lipton, Li (2023) `https://d2l.ai/`
- "Understanding Deep Learning", Prince (2023)
- Stanford, CS230 course `https://stanford.edu/~shervine/teaching/cs-230/`
- "Deep learning with R", `https://srdas.github.io/DLBook/DeepLearningWithR.html`

Thank You!